

Práctica 5 – Desacoplamiento

Descripción de los pasos seguidos para cumplir los objetivos

Actividad 1:

Crear un contenedor con docker que contenga nuestra aplicación que permita comprobar su funcionamiento (e.g. una pagina web)

Para esto, debemos crear nuestra página web(HTML) y nuestro archivo Dockerfile:

index.html > html > body > script > run

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>App CN Practica 5</title>
7  </head>
8  <body>
9      <h1>Resultado del Programa</h1>
10     <p id="result"></p>
11     <p id="finish"></p>
12
13     <script>
14         function fA(DataA) {
15             console.log(DataA);
16             return new Promise(resolve => {
17                 setTimeout(() => {
18                     resolve(DataA + "A");
19                     document.getElementById("result").innerHTML = DataA + "A"
20                 }, 5000);
21             });
22         }
23
24         function fB(DataB) {
25             console.log(DataB);
26             return new Promise(resolve => {
27                 setTimeout(() => {
28                     resolve(DataB + "B");
29                     document.getElementById("result").innerHTML = DataB + "B"
30                 }, 3000);
31             });
32         }
33
34         function fC(DataC) {
35             console.log(DataC);
36             return new Promise(resolve => {
37                 setTimeout(() => {
38                     resolve(DataC + "C");
39                     document.getElementById("result").innerHTML = DataC + "C"
40                 }, 4000);
41             });
42         }
```

```

44  async function run() {
45      let w = 'Inicio: ';
46      let x = await fA(w);
47      let y = await fB(x);
48      let z = await fC(y);
49      document.getElementById("result").innerHTML = z
50      document.getElementById("finish").innerHTML = "TERMINADO..."
51  }
52
53  run();
54
55  </script>
56 </body>
57 </html>

```

```

Dockerfile > ...
1  # Usa una imagen base que contenga un servidor web --> nginx
2  FROM nginx:latest
3
4  # Copia el archivo index.html al directorio donde nginx sirve los archivos estáticos
5  COPY index.html /usr/share/nginx/html/index.html

```

Crear un repositorio en ECR y subir el contenedor creado en el paso 1

1. Primero debemos crear nuestro repositorio ECR Privado en AWS. Solo necesitamos ponerle un nombre.
2. A continuación debemos poner nuestras credenciales en la consola AWS CLI de nuestro ordenador, rellenando el archivo '~/.aws/credentials' con los siguiente encontrado en CLI > AWS Details > Cloud Access:

```

[default] aws_access_key_id=<Access_Key>
aws_secret_access_key=<Secret_Key>
aws_session_token=<Token>

```

3. A continuación debemos seguir los siguientes pasos para verificar nuestro Docker, construir la imagen, añadir el TAG a la imagen y finalmente hacer el PUSH a nuestro repositorio ECR.

Si seleccionamos nuestro Repositorio y le damos a 'View push commands' nos mostrará los pasos aplicados a nuestro repositorio:

1. Retrieve an authentication token and authenticate your Docker client to your registry.

Use the AWS CLI:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 329187668450.dkr.ecr.us-east-1.amazonaws.com
```

Note: if you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch, see the instructions [here](#). You can skip this step if your image has already been built:

```
docker build -t webserver_cn .
```

3. After the build is completed, tag your image so you can push the image to this repository:

```
docker tag webserver_cn:latest 329187668450.dkr.ecr.us-east-1.amazonaws.com/webserver_cn:latest
```

4. Run the following command to push this image to your newly created AWS repository:

```
docker push 329187668450.dkr.ecr.us-east-1.amazonaws.com/webserver_cn:latest
```

Finalmente, queda desplegar el contenedor usando Fargate y comparar la experiencia

1. Primero debemos crear un *'Task Definition'*. Poniéndole su nombre, seleccionando su memoria, añadiéndole los roles de **LabRole** y añadiendo el contenedor, usando la URI de nuestro repositorio ECS.

Task definition configuration

Task definition family [Info](#)

Specify a unique task definition family name.

cn_ecs_task

Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

▼ Infrastructure requirements

Specify the infrastructure requirements for the task definition.

Launch type [Info](#)

Selection of the launch type will change task definition parameters.

☒ AWS Fargate

Serverless compute for containers.

☐ Amazon EC2 Instances

Self-managed infrastructure using Amazon EC2 instances.

OS, Architecture, Network mode

Network mode is used for tasks and is dependent on the compute type selected.

Operating system/Architecture [Info](#)

Linux/X86_64

Network mode [Info](#)

awsvpc

Task size [Info](#)

Specify the amount of CPU and memory to reserve for your task.

CPU

1 vCPU

Memory

2 GB

▼ Task roles - conditional

Task role [Info](#)

A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the [IAM console](#).

LabRole

Task execution role [Info](#)

A task execution IAM role is used by the container agent to make AWS API requests on your behalf. If you don't already have a task execution IAM role created, we can create one for you.

LabRole

Container – 1 [Info](#)

Essential container

Remove

Container details

Specify a name, container image and whether the container should be marked as essential. Each task definition must have at least one essential container.

Name

webserver_cn

Image URI

329187668450.dkr.ecr.us-east-1.amazonaws.com/webse

Essential container

Yes

Private registry [Info](#)

Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.

☐ Private registry authentication

Port mappings [Info](#)

Add port mappings to allow the container to access ports on the host to send or receive traffic. Any changes to port mappings configuration impacts the associated service connect settings.

Container port

80

Protocol

TCP

Port name

webserver_cn-80-tcp

App protocol

HTTP

Remove

Add port mapping

2. A continuación, creamos nuestro cluster usando **FarGate**. Solo es necesario asignarle un nombre y seleccionar en Infrastucture, el uso de AWS Fargate (Serverless)

Cluster configuration

Cluster name

There can be a maximum of 255 characters. The valid characters are letters (uppercase and lowercase), numbers, hyphens, and underscores.

Default namespace - *optional*

Select the namespace to specify a group of services that make up your application. You can overwrite this value at the service level.

▼ Infrastructure Info

Serverless

Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances, or external instances using ECS Anywhere.

☒ AWS Fargate (serverless)

Pay as you go. Use if you have tiny, batch or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

☐ Amazon EC2 Instances

Manual configurations. Use for large workloads with consistent resource demands.

☐ External instances using ECS Anywhere

Manual configurations. Use to add data centre compute.

3. Seguimos añadiendo el 'Task' a nuestro cluster de ECR.
En configuración de despliegue, seleccionamos 'Task' y en Familia nuestro **Task Definition**.

Deployment configuration

Application type [Info](#)
Specify what type of application you want to run.

☐ **Service**
Launch a group of tasks handling a long-running computing work that can be stopped and restarted. For example, a web application.

☒ **Task**
Launch a standalone task that runs and terminates. For example, a batch job.

Task definition
Select an existing task definition. To create a new task definition, go to [Task definitions](#).

☐ **Specify the revision manually**
Manually input the revision instead of choosing from the 100 most recent revisions for the selected task definition family.

Family **Revision**

cn_ecs_task 1 (LATEST)

Desired tasks
Specify the number of tasks to launch.

1

Task group
All tasks with the same task group name are considered as a set when performing spread placement.

Seleccionamos un '**Security Group**' que permita el acceso web -> HTTP(80) desde 0.0.0.0/0

- Finalmente comprobamos que funciona accediendo a el apartado 'Networking' de nuestro 'Task' y seleccionando su IP Pública:



Actividad 2:

Divida su aplicación en tres aplicaciones distintas (fA, fB, fC) y modifíquelas para que cada aplicación le pase su resultado a la siguiente usando colas (e.g. SQS o redis) y desplieguelas en AWS. En el ejemplo anterior 'x' e 'y' se mandarian por colas

1. Primero debemos de crear las SQS para las funciones fA y fB.

Details

Type
Choose the queue type for your application or cloud infrastructure.

☒ **Standard Info**
At-least-once delivery, message ordering isn't preserved

- At-least once delivery
- Best-effort ordering

☐ **FIFO Info**
First-in-first-out delivery, message ordering is preserved

- First-in-first-out delivery
- Exactly-once processing

You can't change the queue type after you create a queue.

Name

fA_Queue

A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (_).

<input type="radio"/>	<u>fA_Queue</u>	Standard
<input type="radio"/>	<u>fB_Queue</u>	Standard

2. En esta actividad se modificará el código de la Actividad 1, para:
 - a. Añadir SDK de AWS

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/aws-sdk/2.1188.0/aws-sdk.min.js"></script>
```

- b. Añadir credenciales de AWS
- c. Añadir métodos de Mandar y Recibir mensajes de las SQS
- d. Implementar estos métodos en las funciones fA y fB


```
index.html > html > body > script > delay
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>App CN Practica 5 - Parte 2</title>
7   <script src="https://cdnjs.cloudflare.com/ajax/libs/aws-sdk/2.1188.0/aws-sdk.min.js"></script>
8 </head>
9 <body>
10   <h1>Resultado del Programa - Usando SQS</h1>
11   <p id="result"></p>
12   <p id="finish"></p>
13
14   <script>
15     function delay(ms) {
16       return new Promise(resolve => setTimeout(resolve, ms));
17     }
18
19     // Set the region for AWS services
20     AWS.config.update({
21       region:
22       accessKeyId:
23       secretAccessKey:
24       sessionToken:
25     });
26
27     // Create an SQS service object
28     const sqs = new AWS.SQS({ apiVersion: '2012-11-05' });
29
30     // URLs of the SQS queues
31     const fALink = 'https://sqs.us-east-1.amazonaws.com/329187668450/fA_Queue';
32     const fBLink = 'https://sqs.us-east-1.amazonaws.com/329187668450/fB_Queue';
33
```

```

34 // Function to send a message to the specified queue
35 async function sendMessage(queueUrl, messageBody) {
36     const params = {
37         MessageBody: messageBody,
38         QueueUrl: queueUrl
39     };
40
41     try {
42         const data = await sqs.sendMessage(params).promise();
43         console.log(`Message sent successfully to ${queueUrl}:`, data.MessageId);
44     } catch (error) {
45         console.error(`Error sending message to ${queueUrl}:`, error);
46     }
47 }
48
49 // Function to receive a message from the specified queue
50 async function receiveMessage(queueUrl) {
51     const params = {
52         QueueUrl: queueUrl,
53         MaxNumberOfMessages: 1,
54         VisibilityTimeout: 0,
55         WaitTimeSeconds: 0
56     };
57
58     try {
59         const result = await sqs.receiveMessage(params).promise();
60         return result.Messages ? result.Messages[0].Body : null;
61     } catch (error) {
62         console.error(`Error receiving message from ${queueUrl}:`, error);
63         return null;
64     }
65 }
66
67 async function fA(DataA) {
68     console.log(DataA);
69     const resultA = DataA + "A";
70     await sendMessage(fALink, resultA);
71     //ESPERAR
72     delay(5000);
73     let resultSQSa = await receiveMessage(fALink);
74     document.getElementById("result").innerHTML = resultSQSa;
75     return resultSQSa;
76 }

```

```

78     async function fB(DataB) {
79         console.log(DataB);
80         const resultB = DataB + "B";
81         await sendMessage(fBLink, resultB);
82         //ESPERAR
83         delay(3000);
84         let resultSQSb = await receiveMessage(fBLink);
85         document.getElementById("result").innerHTML = resultSQSb;
86         return resultSQSb;
87     }
88
89     async function fC(DataC) {
90         console.log(DataC);
91         const resultC = DataC + "C";
92         //ESPERAR
93         delay(4000);
94         document.getElementById("result").innerHTML = resultC;
95         return resultC;
96     }
97
98     async function run() {
99         let w = 'Inicio: ';
100         document.getElementById("result").innerHTML = w;
101         let x = await fA(w);
102         let y = await fB(x);
103         let z = await fC(y);
104         document.getElementById("result").innerHTML = z;
105         document.getElementById("finish").innerHTML = "TERMINADO...";
106     }
107
108     run();
109 </script>
110 </body>
111 </html>

```

3. Se añadirá a un ECR, ECS y se desplegará para comprobar que funciona:



Actividad 3:

Configure un 'topic' en AWS SNS para que fA se subscriba y tome su dato de entrada de ahi. En el ejemplo anterior, fA recibiría 'w' de un topic.

1. Primero debemos de crear el **Topic** y **Subscripción** de nuestro SNS:

1.1. Configuramos un **Topic** 'Standard' con un nombre cualquiera.

Type [Info](#)
Topic type cannot be modified after topic is created

☐ **FIFO (first-in, first-out)**

- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

☒ **Standard**

- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

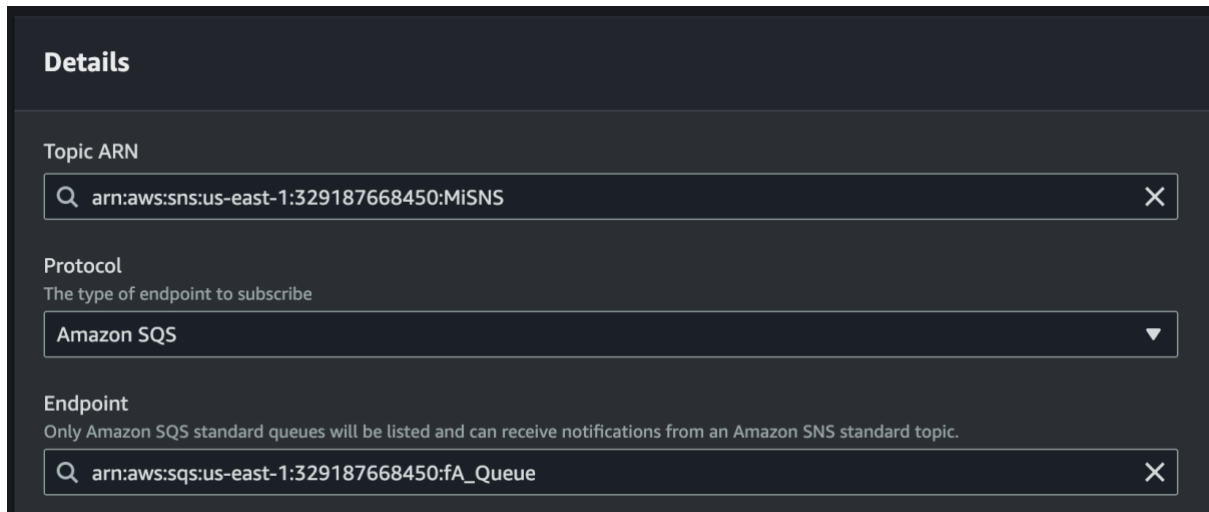
Name

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - optional [Info](#)
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

Maximum 100 characters.

1.2 Luego crearemos una suscripción donde seleccionaremos nuestro **Topic**, **SQS** como protocolo y nuestro Queue de A como Endpoint:



The screenshot shows the 'Details' section of an Amazon SNS subscription. It contains three main fields:

- Topic ARN:** A text input field containing 'arn:aws:sns:us-east-1:329187668450:MiSNS'.
- Protocol:** A dropdown menu with the text 'The type of endpoint to subscribe' and the selected option 'Amazon SQS'.
- Endpoint:** A text input field containing 'arn:aws:sqs:us-east-1:329187668450:fA_Queue'. Below this field is a note: 'Only Amazon SQS standard queues will be listed and can receive notifications from an Amazon SNS standard topic.'

2. Para esta Actividad necesitaremos crear 3 programas diferentes, en nuestro caso tendremos 3 carpetas diferentes que cada contiene su *index.html* y su *Dockerfile*.

Programa fA:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>fA - Publicador de SNS y Consumidor de SQS</title>
</head>
<body>
  <h1>(fA) Resultado del Programa - Usando SQS</h1>
  <h4 id="result"></h4>
  <h4 id="end"></h4>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/aws-sdk/2.1188.0/aws-sdk.min.js"></script>

  <script>
    if (
      !localStorage.getItem('accessKeyId') &&
      !localStorage.getItem('secretAccessKey') &&
      !localStorage.getItem('sessionToken')
```

```

){
  // Pedir al usuario que ingrese el Access Key ID
  const accessKeyId = prompt("Ingresa el Access Key ID de AWS:");

  // Pedir al usuario que ingrese el Secret Access Key
  const secretAccessKey = prompt("Ingresa el Secret Access Key de AWS:");

  // Pedir al usuario que ingrese el Session Token (opcional, dependiendo del caso)
  const sessionToken = prompt("Ingresa el Session Token de AWS (si aplica):");

  localStorage.setItem('accessKeyId', accessKeyId)
  localStorage.setItem('secretAccessKey', secretAccessKey)
  localStorage.setItem('sessionToken', sessionToken)
}

AWS.config.update({
  region: 'us-east-1',
  accessKeyId: localStorage.getItem('accessKeyId'),
  secretAccessKey: localStorage.getItem('secretAccessKey'),
  sessionToken: localStorage.getItem('sessionToken')
})

const sqs = new AWS.SQS({ apiVersion: '2012-11-05' })
const sqs_FA = 'https://sqs.us-east-1.amazonaws.com/329187668450/fA_Queue'
const topicArn = 'arn:aws:sns:us-east-1:329187668450:MiSNS'; // Reemplaza con el ARN de tu tema
const message = 'Inicio: ';
const sns = new AWS.SNS({ apiVersion: '2010-03-31' });

// Publicar el mensaje al tema SNS
async function publishMessage() {
  const params = {
    Message: message,
    TopicArn: topicArn
  };

  try {
    const data = await sns.publish(params).promise();
    console.log("Mensaje publicado al tema SNS:", data);
  } catch (error) {
    console.error("Error al publicar el mensaje al tema SNS:", error);
  }
}

```

```

    }
}

async function sendMessage(queueURL, messageBody) {
    const params = {
        DelaySeconds: 0, //se juega con esto en cada aplicacion para los delay
        MessageBody: messageBody,
        QueueUrl: queueURL
    };
    try {
        const data = await sqs.sendMessage(params).promise();
    } catch (error) {
        console.log(error)
    }
}

async function receiveMessage(queueURL) {
    const params = {
        QueueUrl: queueURL,
        MaxNumberOfMessages: 1,
        VisibilityTimeout: 0,
        WaitTimeSeconds: 0 //se juega con esto en cada aplicacion para los delay
    };
    try {
        const sol = await sqs.receiveMessage(params).promise();
        return sol.Messages ? sol.Messages[0].Body : null;
    } catch (error) {
        console.log(error)
    }
}

function wait(ms) {
    return new Promise(resolve => setTimeout(resolve, ms))
}

function esJSON(cadena) {
    try {
        JSON.parse(cadena);
        return true;
    } catch (error) {

```

```

        return false;
    }
}

localStorage.setItem('sessionA',false)
localStorage.setItem('sessionB', false)

async function fA() {
    // Espera el mensaje del tema SNS
    let snsMessage = await receiveMessage(sqs_FA);
    if(esJSON(snsMessage)){
        const RsnsMessage = JSON.parse(snsMessage).Message;
        const A = RsnsMessage + "A";
        await sendMessage(sqs_FA, A);
        await wait(5000)
        await localStorage.setItem('sessionA',true)
        return A;
    }else{
        const A = snsMessage + "A";
        await sendMessage(sqs_FA, A);
        await wait(5000)
        await localStorage.setItem('sessionA',true)
        return A ;
    }
}

async function run() {
    try {
        await publishMessage()
        let x = await fA();
        document.getElementById('result').innerHTML = x;
        document.getElementById('end').innerHTML = "TERMINADO...";
    } catch (error) {
        console.error(error);
    }
}

run()
</script>
</body>

```



```
</html>
```

Programa fB:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>FB - Consumidor de SQS</title>
</head>
<body>
  <h1>(fB) Resultado del Programa - Usando SQS</h1>
  <h4 id="result"></h4>
  <h4 id="end"></h4>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/aws-sdk/2.1188.0/aws-sdk.min.js"></script>

  <script>
    if (
      !localStorage.getItem('accessKeyId') &&
      !localStorage.getItem('secretAccessKey') &&
      !localStorage.getItem('sessionToken')
    ) {
      // Pedir al usuario que ingrese el Access Key ID
      const accessKeyId = prompt("Ingresa el Access Key ID de AWS:");

      // Pedir al usuario que ingrese el Secret Access Key
      const secretAccessKey = prompt("Ingresa el Secret Access Key de AWS:");

      // Pedir al usuario que ingrese el Session Token (opcional, dependiendo del caso)
      const sessionToken = prompt("Ingresa el Session Token de AWS (si aplica):");

      localStorage.setItem('accessKeyId', accessKeyId)
      localStorage.setItem('secretAccessKey', secretAccessKey)
      localStorage.setItem('sessionToken', sessionToken)
    }

    AWS.config.update({
      region: 'us-east-1',
```

```

    accessKeyId: localStorage.getItem('accessKeyId'),
    secretAccessKey: localStorage.getItem('secretAccessKey'),
    sessionToken: localStorage.getItem('sessionToken')
  })

const sqs = new AWS.SQS({ apiVersion: '2012-11-05' })
const sqs_FB = 'https://sqs.us-east-1.amazonaws.com/329187668450/fB_Queue';
const sqs_FA = 'https://sqs.us-east-1.amazonaws.com/329187668450/fA_Queue';

async function sendMessage(queueURL, messageBody) {
  const params = {
    MessageBody: messageBody,
    QueueUrl: queueURL,
    DelaySeconds: 3,
  };
  try {
    const data = await sqs.sendMessage(params).promise();
  } catch (error) {
    console.log(error)
  }
}

function esJSON(cadena) {
  try {
    JSON.parse(cadena);
    return true;
  } catch (error) {
    return false;
  }
}

async function receiveMessage(queueURL) {
  const params = {
    QueueUrl: queueURL,
    MaxNumberOfMessages: 1,
    VisibilityTimeout: 0,
    WaitTimeSeconds: 0
  };
  try {
    const sol = await sqs.receiveMessage(params).promise();
    return sol.Messages ? sol.Messages[0].Body : null;
  }
}

```

```

    } catch (error) {
        console.log(error)
    }
}

async function fB() {
    console.log("Esperando en B - sessionA:", localStorage.getItem('sessionA'));

    while (localStorage.getItem('sessionA') !== 'false') {
        await wait(1000);
        console.log("Esperando en B - sessionA:", localStorage.getItem('sessionA'));
    }

    let sqsResultA = await receiveMessage(sqs_FA)
    if (esJSON(sqsResultA)) {
        const RsnsMessage = JSON.parse(sqsResultA).Message;
        const B = RsnsMessage + "B"
        await sendMessage(sqs_FB, B)
        await wait(3000);
        await localStorage.setItem('sessionB', true)
        return B
    } else {
        const B = sqsResultA + "B"

        await sendMessage(sqs_FB, B)
        await wait(3000);
        await localStorage.setItem('sessionB', true)
        return B
    }
}

function wait(ms) {
    return new Promise(resolve => setTimeout(resolve, ms))
}

async function run() {
    try {
        let y = await fB();
        document.getElementById('result').innerHTML = y;
        document.getElementById('end').innerHTML = "TERMINADO...";
    }
}

```

```

        } catch (error) {
            console.error(error);
        }
    }

    run()
</script>
</body>
</html>

```

Programa fC:

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>FC - Consumidor de SQS</title>
</head>
<body>
    <h1>(fC) Resultado del Programa - Usando SQS</h1>
    <h4 id="result"></h4>
    <h4 id="end"></h4>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/aws-sdk/2.1188.0/aws-sdk.min.js"></script>

    <script>
        if (
            !localStorage.getItem('accessKeyId') &&
            !localStorage.getItem('secretAccessKey') &&
            !localStorage.getItem('sessionToken')
        ) {
            // Pedir al usuario que ingrese el Access Key ID
            const accessKeyId = prompt("Ingresa el Access Key ID de AWS:");

            // Pedir al usuario que ingrese el Secret Access Key
            const secretAccessKey = prompt("Ingresa el Secret Access Key de AWS:");

            // Pedir al usuario que ingrese el Session Token (opcional, dependiendo del caso)
            const sessionToken = prompt("Ingresa el Session Token de AWS (si aplica):");

```

```

    localStorage.setItem('accessKeyId', accessKeyId)

    localStorage.setItem('secretAccessKey', secretAccessKey)

    localStorage.setItem('sessionToken', sessionToken)
}

AWS.config.update({
    region: 'us-east-1',
    accessKeyId: localStorage.getItem('accessKeyId'),
    secretAccessKey: localStorage.getItem('secretAccessKey'),
    sessionToken: localStorage.getItem('sessionToken')
})

const sqs = new AWS.SQS({ apiVersion: '2012-11-05' })
const sqs_FB = 'https://sqs.us-east-1.amazonaws.com/329187668450/fB_Queue';

async function receiveMessage(queueURL) {
    const params = {
        QueueUrl: queueURL,
        MaxNumberOfMessages: 1,
        VisibilityTimeout: 0,
        WaitTimeSeconds: 0
    };
    try {
        const sol = await sqs.receiveMessage(params).promise();
        return sol.Messages ? sol.Messages[0].Body : null;
    } catch (error) {
        console.log(error)
    }
}

function esJSON(cadena) {
    try {
        JSON.parse(cadena);
        return true;
    } catch (error) {
        return false;
    }
}

async function fC() {
    console.log("Esperando en C - sessionB:", localStorage.getItem('sessionB'));

```

```

while (localStorage.getItem('sessionB')=='false') {
    await wait(1000);
    console.log("Esperando en C - sessionB:", localStorage.getItem('sessionB'));
}

let sqsResultB = await receiveMessage(sqs_FB)
if(esJSON(sqsResultB)){
    const RsnsMessage = JSON.parse(sqsResultB).Message;
    const C = RsnsMessage+"C"
    await wait(4000)
    return C
}else{
    const C = sqsResultB+"C"
    await wait(4000)
    return C
}
}

function wait(ms) {
    return new Promise(resolve => setTimeout(resolve, ms))
}

async function run() {
    try {
        let z = await fC();
        document.getElementById('result').innerHTML = z;
        document.getElementById('end').innerHTML = "TERMINADO...";
    } catch (error) {
        console.error(error);
    }
}

run()
</script>
</body>
</html>

```

Tras ejecutar nuestros 3 programas veremos que el resultado es el siguiente (IMPORTANTE: introducir las credenciales en el LocalStorage):

(fA) Resultado del Programa - Usando SQS

Inicio:A

TERMINADO...

(fB) Resultado del Programa - Usando SQS

Inicio:AB

TERMINADO...

(fC) Resultado del Programa - Usando SQS

Inicio:ABC

TERMINADO...

Presupuesto y estimación de gasto de los recursos desplegados

⌵ Cost summary [Info](#)

Month-to-date cost

\$0.09

↑ 119% compared to last month for same period

Last month's cost for same time period

\$0.04

Nov 1 – 16

Total forecasted cost for current month

\$0.13

↑ 97% compared to last month's total costs

Last month's total cost

\$0.07

Nov 2023

<div></div> Amazon Elastic Container Service	0.008
<div></div> EC2 - Other	0.004
<div></div> AmazonCloudWatch	0.011
<div></div> CloudWatch Events	0
<div></div> Amazon Simple Storage Service	0
<div></div> Others	0.045

Dec 2023



<div></div> Amazon Elastic Container Service	0.087
<div></div> EC2 - Other	0.005
<div></div> AmazonCloudWatch	0.001
<div></div> CloudWatch Events	0
<div></div> Amazon Simple Storage Service	0
<div></div> Others	0